XL CILAMCE
IBERO-LATIN AMERICAN CONGRESS ON COMPUTATIONAL METHODS IN ENGINEERING
NOVEMBER 11-14, 2019
Praiamar Natal Hotel & Convention
Natal, RN-BRAZIL

A COMPARATIVE STUDY OF GLOBAL AND LOCAL PRECONDITIONERS FOR PARALLEL FINITE ELEMENT COMPUTATIONS

Leonardo Muniz de Lima
lmuniz@ifes.edu.br
Instituto Federal do Espírito Santo
Av. Morobá, 248 - Morobá, Aracruz, 29192-733, ES, Brazil
Andrea M. P. Valli
avalli@inf.ufes.br
Lucia Catabriga
luciac@inf.ufes.br
Universidade Federal do Espírito Santo
A. Fernando Ferrari, 514 - Goiabeiras, Vitória, 29075-910, ES, Brazil

Abstract. Numerous engineering problems require an enormous computational effort to solve large sparse linear systems with matrices resulting of finite element discretizations. Parallel computing of GMRES or other Krylov subspace-based method combined with a good preconditioner is an excellent option to fulfill this role. Mostelly, the two main operations of Krylov subspace methods, the inner and the matrix-vector products, demand less memory consumption and present better parallelism when compared with the main operations of the traditional direct solvers. Global storage allows the use of block Jacobi preconditioners based on incomplete LU factorization. In contrast, local storage schemes provide preconditioners performed at the element level factorizations (e.g., the local LU, the local Gauss-Seidel and the diagonal or the block-diagonal schemes). In this work, we present a trade-off analysis between global and local preconditioners for parallel finite element implementations. In the context of a specific domain decomposition approach, we evaluate the preconditioners according to memory usage, load balancing and MPI communication. Robustness and scalability of these parallel preconditioning strategies are demonstrated for two benchmark cases: the rotating cone modeled by the transient transport equation, and the explosion problem modeled by the Euler equations. All experiments point out the supremacy of the global preconditioners, and the local preconditioners dependence on the number of degrees of freedom per node.

Keywords: Finite Element Method, GMRES, Parallel Global and Local Preconditioners

1 Introduction

In practical applications, it is mandatory to exploit parallelism in combination with suitable solution techniques to solve very large and sparse systems arising from finite element discretizations [1]. They can be solved by either sparse direct solvers or by iterative methods. In the last decates, clever variations of Gauss elimination have been tuned to perfection by finding strategies for reducing the bandwidth of matrices to guarantee stable LU decompositions and small fill-in in the triangular factors of the elimination [2]. On the other hand, today the iterative methods that are applied for solving large-scale linear systems are mostly preconditioned Krylov subspace solvers. Any of them may be a good option if it converges after only a few iterations [3]. That might be the case if a good preconditioner is applied. So, many efforts have been made to achieve this goal. Preconditioned Krylov space methods for solving large sparse systems have the special feature that the matrix needs only be given as an operator: one must be able to compute matrix-vector products. This operation include the application of a preconditioner, which also requires the solution of a large linear system. In this context, the Generalized Minimal Residual (GMRES) [4] method has become a suitable option since the two main operations of Krylov subspace methods, the inner and the matrix-vector products, demand less memory consumption and present better parallelism when compared with the main operations of the traditional direct solvers. The great question to ask is whether or not it is possible to find preconditioning techniques that have a high degree of parallelism as well good intrinsic qualities [5].

Manguoglu et al. [6] showed that global preconditioners based on narrowband linear systems [7], such as SPIKE [8], present robustness and scalability. In our work [9], we suggested an alternative arrangement to use SPIKE that avoids excessive overhead and allows solving finite element applications indeed in parallel, applying combinatorial techniques just once and in the beginning of the process. After the proper preprocessing is established, we always work in parallel, from reading finite element data, through the resolution of linear systems using preconditioners to post-processing. We also established a domain decomposition that provides scalability and better convergence properties [9, 10]. For finite element problems, our domain decomposition for narrowband systems [9] enables, with minimal adjustments, parallelization of global ILU preconditioners based on Compressed Storage Row (CSR) format [5] and local preconditioners based on element-by-element (EBE) storage. However, we need combinatorial techniques to preprocess the original finite element linear systems that are not narrowband. In our work [10], several combinatorial techniques using SPIKE preconditioner are analyzed. Based on this work, we selected the reverse Cuthill-McKee (RCM) [11] reordering method to obtain the narrowband system in the preprocessing finite element model preparation.

The objective of this work is to present a trade-off analysis between global and local preconditioners for parallel finite element implementations. In our previous work [12], the efficiency of sequential local preconditioners for 2D finite element transport and Euler problems were presented for different types of data structures. We compared them with the most common sequential incomplete LU factorization, that is, the well-known ILUp [13], where p represents the number of fill-in levels admitted. Here, outstanding parallel versions of element-by-element representatives proposed by Muller et al. [12] are provided for transport equation: a diagonal preconditioner (DIAGe), an LU-factorization preconditioner (LUe) and a Gauss-Seidel preconditioner (SGSe); and for Euler Equations: a block diagonal preconditioner (BlockDIAGe), a block LU-factorization preconditioner (BlockLUe) and a block Gauss-Seidel preconditioner (BlockSGSe). The goal is to compare them with a parallel global block Jacobi preconditioner based on incomplete LU factorization in CSR format. All global and local preconditioner versions are adapted to the same domain decomposition context used in our global parallel preconditioner SPIKE [9, 10]. The remainder of this text is organized as follows. Section 2 briefly describes the finite element formulations and time advance algorithm. The global and local parallel preconditioners are addressed in Section 3. Numerical experiments and conclusions are presented in Sections 4 and 5, respectively.

2 Finite Element Formulations and Time Advance Algorithm

Let us consider two class of problems characterized by the transport and Euler equations. Transport equation describes quantities to be transported, for example, temperature and concentration. Euler equations represent a system of conservative laws governed by inviscid compressible flows. To obtain the finite element discretization, we consider $\mathcal{T}_h$ a set of triangular partitions of the domain Ω into n_{el} elements such that

$$\mathcal{T}_h = \{\Omega_e \,|\, \Omega = \cup_{e=1}^{n_{el}} \Omega_e,\, \Omega_i \cap \Omega_j = \emptyset,\, i \neq j,\, i,j = 1,\ldots,n_{el}\}. \tag{1}$$

After the discretization of the governing equations in space, the resulting time discrete equations are solved using a predictor-multi-corrector algorithm. The details of both transport and Euler finite element stabilized formulations are shown in the next two sub-sections. Later, the time advancing strategy used on both equations is presented.

2.1 Transport Equation

The transport equation may be expressed by

$$\begin{aligned} \frac{\partial u}{\partial t} - \nabla\cdot(\boldsymbol{\kappa}\nabla u) + \boldsymbol{\beta}\cdot\nabla u + \sigma u &= f, \;\; \text{in } \Omega\times(0,t_f), \\ u &= g, \;\; \text{on } \Gamma\times(0,t_f), \\ u(\mathbf{x},0) &= u_0(\mathbf{x}), \;\; \text{in } \Omega, \end{aligned} \tag{2}$$

where the functions $g(\mathbf{x},t)$ and $u_0(\mathbf{x}) = u(\mathbf{x},0)$ are the boundary and initial conditions, respectively. The variables are described as follows: u is the quantity being transported; $\Omega \subset \mathbb{R}^2$ is the domain of the problem and $\Gamma = \partial\Omega$ is its boundary; $t_f > 0$ is the final time; $\boldsymbol{\kappa}$ represents the diffusivity tensor; $\boldsymbol{\beta}$ is the velocity field, given by $\boldsymbol{\beta} = (\beta_x, \beta_y)$; σ is the reaction coefficient; and f represents the source term. Also, consider the two finite element spaces, $\mathcal{V}_h$ (trial) and $\mathcal{S}_h$ (weighting), defined by

$$\begin{aligned} \mathcal{V}_h &= \{v_h \in H_0^1 \,|\, v|_{\Omega_e} \in \mathbb{P}_1(\Omega_e),\, \forall\Omega_e \in \mathcal{T}_h \text{ and } v|_{\Gamma_D} = 0\}, \\ \mathcal{S}_h &= \{u_h = u_h(\cdot,t) \in H^1 \,|\, t \in [0,t_f],\, u_h(\mathbf{x},t) = g, \forall\mathbf{x} \in \Gamma\}, \end{aligned}$$

where $\mathbb{P}_1(\Omega_e)$ is the set of first order polynomial functions, defined in Ω_e, and Γ_D is the Dirichlet part of the boundary. Thereby, the finite element stabilized formulation reads: given g and f, find $u \in \mathcal{S}_h$ such that, for all $v_h \in \mathcal{V}_h$,

$$\int_\Omega \left(v_h \frac{\partial u_h}{\partial t} + \boldsymbol{\kappa}\nabla u_h \nabla v_h + (\boldsymbol{\beta}\cdot\nabla u_h)\, v_h + \sigma u_h v_h \right) d\Omega + \sum_{e=1}^{n_{el}} \int_{\Omega_e} R(u_h)\, \tau\, \boldsymbol{\beta}\cdot\nabla v_h \, d\Omega_e \;\; + \\ \sum_{e=1}^{n_{el}} \int_{\Omega_e} \nabla v_h \, \boldsymbol{\delta} \, \nabla u_h \, d\Omega_e = \int_\Omega f v_h \, d\Omega + \int_\Gamma g v_h \, ds, \tag{3}$$

where $R(u_h)$ is the residual of Eq. (2), defined by

$$R(u_h) = \frac{\partial u_h}{\partial t} + \boldsymbol{\kappa}\nabla u_h + (\boldsymbol{\beta}\cdot\nabla u_h) + \sigma u_h - f,$$

τ is the stabilization parameter for SUPG [14] and $\boldsymbol{\delta}$ is the discontinuity-capturing operator YZβ [15]. The SUPG parameter is given as follows

$$\tau = \frac{h\xi}{2||\,\boldsymbol{\beta}\,||_\infty}, \;\; \text{with } \xi = \max\left\{0, 1 - \frac{1}{Pe}\right\} \;\; \text{and} \;\; Pe = \frac{||\,\boldsymbol{\beta}\,||_\infty h}{2\boldsymbol{\kappa}} \;\; \text{(Peclet number)}.$$

The discontinuity-capturing operator YZβ is defined as $\boldsymbol{\delta} = \delta \boldsymbol{I}$, where

$$\delta = |Y^{-1}R| \left(\sum_{i=1}^{d} \left| Y^{-1} \frac{\partial u_h}{\partial x_i} \right|^2 \right)^{\frac{\beta}{2}-1} \left(\frac{\tilde{h}}{2} \right)^{\beta},$$

with

$$\tilde{h} = 2 \left(\sum_{a=1}^{n_{el}} |\mathbf{j} \cdot \nabla \boldsymbol{N_a}| \right)^{-1} \quad \text{and} \quad \mathbf{j} = \frac{\nabla u_h}{|| \nabla u_h ||},$$

and $Y = u_{ref}$ is the reference value of the scalar field u_h [16].

2.2 Euler Equations

The Euler equations model a system of conservation laws governing inviscid compressible flows. For two-dimensional domains, the conservative form of the Euler equations with no source terms in given by

$$\frac{\partial \mathbf{U}}{\partial t} + \frac{\partial \mathbf{F}_x}{\partial x} + \frac{\partial \mathbf{F}_y}{\partial y} = \mathbf{0}, \text{ on } \Omega \times [0, t_f], \tag{4}$$

where $\mathbf{U} = (\rho, \rho v_x, \rho v_y, \rho e)$, ρ is the fluid density, $\mathbf{v} = (v_x, v_y)^T$ is the velocity field, e represents the total energy per unit mass, $\mathbf{F}_x$ and $\mathbf{F}_y$ represents the Euler fluxes. Another way to represent the Euler equations is to write Eq. (4) in terms of the Jacobian matrices $\mathbf{A}_x = \frac{\partial \mathbf{F}_x}{\partial \mathbf{U}}$ and $\mathbf{A}_x = \frac{\partial \mathbf{F}_y}{\partial \mathbf{U}}$. Thus,

$$\frac{\partial \mathbf{U}}{\partial t} + \mathbf{A}_x \frac{\partial \mathbf{U}}{\partial x} + \mathbf{A}_y \frac{\partial \mathbf{U}}{\partial x} = \mathbf{0}. \tag{5}$$

We consider the following boundary conditions associated with Eq. (5)

$$\begin{aligned} \mathbf{BU} &= \mathbf{G}, \text{ on } \Gamma \times [0, t_f], \\ \mathbf{U}(\mathbf{x}, 0) &= \mathbf{U}_0, \text{ on } \Omega, \end{aligned}$$

with given $\mathbf{G} = (g_1(t), g_2(t), g_3(t), g_4(t))^T$, $\mathbf{U}_0$ and a general boundary condition operator $\mathbf{B}$.

Now, consider the following two new finite dimensional spaces $\mathbf{S}^h$ and $\mathbf{V}^h$ given by

$$\begin{aligned} \mathbf{S}^h &= \left\{ \mathbf{U}^h \,|\, \mathbf{U}^h \in [\mathbf{H}^{1h}(\Omega)]^4,\, \mathbf{U}^h|_{\Omega_e} \in [\mathbb{P}_1(\Omega_e)]^4,\, \mathbf{U}^h \cdot \mathbf{e}_k = g_k(t) \text{ on } \Gamma_{g_k} \right\}, \\ \mathbf{V}^h &= \left\{ \mathbf{W}^h \,|\, \mathbf{W}^h \in [\mathbf{H}^{1h}(\Omega)]^4,\, \mathbf{W}^h|_{\Omega_e} \in [\mathbb{P}_1(\Omega_e)]^4,\, \mathbf{W}^h \cdot \mathbf{e}_k = 0 \text{ on } \Gamma_{g_k} \right\}, \end{aligned}$$

where $k = 1, \ldots, 4$ and $\mathbf{H}^{1h}(\Omega) \subset \mathbf{C}^0(\Omega)$ is the finite dimensional function space over Ω. The Euler finite element stabilized formulation is: find $\mathbf{U}^h \in \mathbf{S}^h$ such that, for all $\mathbf{W}^h \in \mathbf{V}^h$

$$\begin{aligned} \int_\Omega \mathbf{W}^h \cdot \left(\frac{\partial \mathbf{U}^h}{\partial t} + \mathbf{A}_x^h \frac{\partial \mathbf{U}^h}{\partial x} + \mathbf{A}_y^h \frac{\partial \mathbf{U}^h}{\partial y} \right) d\Omega \quad & + \\ \sum_{e=1}^{n_{el}} \int_{\Omega_e} \boldsymbol{\tau} \mathbf{A}_i^h \left(\frac{\partial \mathbf{W}^h}{\partial x_i} \right) \cdot R(\mathbf{U}^h)\, d\Omega \quad & + \\ \sum_{e=1}^{n_{el}} \int_{\Omega_e} \boldsymbol{\delta} \left(\frac{\partial \mathbf{W}^h}{\partial x} \frac{\partial \mathbf{U}^h}{\partial x} + \frac{\partial \mathbf{W}^h}{\partial y} \frac{\partial \mathbf{U}^h}{\partial y} \right) d\Omega \quad &= \quad \mathbf{0},\ i = 1, 2, \end{aligned} \tag{6}$$

where $R(\mathbf{U}^h)$ is the residual vector, as

$$R(\mathbf{U}^h) = \frac{\partial \mathbf{U}^h}{\partial t} + \mathbf{A}_x^h \frac{\partial \mathbf{U}^h}{\partial x} + \mathbf{A}_y^h \frac{\partial \mathbf{U}^h}{\partial y},$$

and the associated SUPG stabilized parameter is given by $\boldsymbol{\tau} = \tau\mathbf{I}$ such that

$$\tau = \max\left\{0, \tau_t + \zeta(\tau_a - \tau_\delta)\right\},$$

with

$$\zeta = \frac{2\alpha CFL}{1+2\alpha CFL}, \quad \tau_t = \frac{2}{3(1+2\alpha CFL)}\tau_a, \quad \tau_a = \frac{h}{2(c+|\,\mathbf{v}\boldsymbol{\beta}\,|)}, \quad \tau_\delta = \frac{\delta}{(c+|\,\mathbf{v}\boldsymbol{\beta}\,|)^2}. \tag{7}$$

The CFL parameter corresponds to the Courant-Friedrichs-Levy number, and it is defined here as

$$CFL = \frac{(c+|\,\mathbf{v}\boldsymbol{\beta}\,|)\Delta t}{h}, \quad \text{with } \boldsymbol{\beta} = \frac{\nabla||\,\mathbf{U}\,||_2^2}{||\,\nabla||\,\mathbf{U}\,||_2^2\,||_2}.$$

In Eq. (7), c represents the acoustic speed, h is the mesh parameter, α controls stability and accuracy of the time-marching scheme and δ is the discontinuity-capturing parameter. In Eq.(6), $\boldsymbol{\delta}$ is the discontinuity-capturing operator YZβ given by $\boldsymbol{\delta} = \delta\mathbf{I}$ such that

$$\delta = ||\,\mathbf{Y}^{-1}R(\mathbf{U}^h)\,|| \left(\sum_{i=1}^{d}\left|\left|\mathbf{Y}^{-1}\frac{\partial\mathbf{U}^h}{\partial x_i}\right|\right|^2\right)^{\frac{\beta}{2}-1} \left|\left|\mathbf{Y}^{-1}\mathbf{U}^h\right|\right|^{1-\beta}\left(\frac{\tilde{h}}{2}\right),$$

where $\mathbf{Y}$ is the following matrix

$$\mathbf{Y} = \begin{bmatrix} U_1|_{ref} & 0 & 0 & 0 \\ 0 & U_2|_{ref} & 0 & 0 \\ 0 & 0 & U_3|_{ref} & 0 \\ 0 & 0 & 0 & U_4|_{ref} \end{bmatrix},$$

and $U_i|_{ref}, i = 1, \cdots, 4$, are the reference values for $\mathbf{U}$ [16].

2.3 Predictor-multi-corrector

The spatial discretization of the variational formulations, Eq. (3) for transport and Eq. (6) for Euler, lead to a set of coupled non-linear ordinary differential equations, that can be represented by $\mathbf{Ma} + \mathbf{C}(\mathbf{v}) = \mathbf{F}$, where $\mathbf{v}$ is the vector of nodal values of u for transport (or $\mathbf{U}$ for Euler), and $\mathbf{a}$ is the time derivative of $\mathbf{v}$. Here, $\mathbf{M}$, $\mathbf{C}(\mathbf{v})$ and $\mathbf{F}$ are the resulting finite element spatial discretization structures, with the residual vector defined as $\mathbf{R} = \mathbf{F} - \mathbf{Ma} - \mathbf{Cv}$. To perform the time advancing, we adapted the implicit predictor-multi-corrector algorithm presented in [17] to our parallel approach (see Algorithm 1). Basically, there is a main loop to advance in time, a prediction phase and inner multi-correction loops performed on each partition i. In the algorithm, the superscripts n and j are related to time and multi-corrector counters, respectively. Lines of MPI syncronizations with the parallel operations are highlighted. For each multi-correction, finite element matrices are calculated and assembled, preconditioners are set up (see Section 3), and a GMRES solutions are required. Besides, scaling operations are performed just for LU and Gauss-Seidel local preconditioners (see Subsection 3.2). More details can be found in [18].

3 Finite Element Preconditioning

The main idea behind a good preconditioner is to obtain an appropriate matrix M such that $M^{-1}A$ is well-conditioned [5], where A is the coefficient matrix resulting from the finite element formulation. Theoretically, the best choice for M is A, but to find A^{-1} is harder than to solve the system $AX = F$. In

Algorithm 1 Parallel predictor-multi-corrector.

do
- **Predictor phase:**
- $j \leftarrow 0$
- $v_i^{n+1,0} \leftarrow v_i^n + (1-\lambda)\Delta t a_i^n$
- $a_i^{n+1,0} \leftarrow 0$
- **Multi-corrector phase:**
- **do**
  - ***MPI Update*** of $v_i^{n,j}$
  - ***MPI Update*** of $a_i^{n,j}$
  - Calculation and assembly of matrix $M_i^* = \mathbf{M} + \lambda\Delta t\mathbf{C}$ and vector $R_i^{n,j}$ from each elementary A^e
  - Scaling of matrix M_i^* and vector $R_i^{n,j}$
  - Setup of the preconditioner
  - Solve $M_i^*\Delta a_i = R_i^{n,j}$ using the ***parallel preconditioned GMRES***
  - $v_i^{n,j+1} \leftarrow v_i^{n,j} + \lambda\Delta t\Delta a_i$
  - $a_i^{n,j+1} \leftarrow a_i^{n,j} + \Delta a_i$
  - Apply the ***parallel inner product*** to obtain the norm $|a_i^{n,j}|$
  - Apply the ***parallel inner product*** to obtain the norm $|\Delta a_i|$
  - $j \leftarrow j+1$
- **while** *($j < j_{MAX}$ .AND. $\frac{|\Delta a|}{|a|} > tol$)*
- $t_{n+1} \leftarrow t_n + \Delta t$

while *($t_n \leq t_{final}$)*

practice, instead of determining M^{-1} explicitly, the preconditioning is defined from matrix-vector products and through resolution of simple linear systems involving M. Since all preconditioners considered here are conceived from our parallel version of SPIKE [10], a preprocessing of the coefficient matrix should be employed, transforming the original sparse matrix A into a banded matrix.

For all parallel preconditioners, we use the Reverse Cuthill-McKee (RCM) reordering scheme as a preprocessing stage to obtain the banded matrix A. They promote a suitable matrix rearrangement such that a chosen central band produces a good preconditioner for Krylov iterative methods. Nevertheless, for problems whose assembly and reassembly of the finite element matrices are numerous, such modifications can become almost infeasible to the parallel processing due to computational demand. Thus, still in the preprocess, we use a suitable domain decomposition assuring all phases of the parallel finite element computation occurs indeed in parallel [9]: from the finite element data mesh reading through the finite element matrices assembling to the preconditioned linear system solutions.

We established a domain decomposition that provides better scalability and convergence properties. That is, the banded matrix A is divided as shown in Eq. (8), with diagonal blocks A_i ($i = 1, 2, ..., p$) and coupling blocks B_i ($i = 1, 2, \cdots, p-1$) and C_i ($i = 2, ..., p$), where p is the number of partitions. In general, the preconditioning matrix M is formed using these blocks or sub-matrices of them in different ways, and their construction depend on the data structure used.

$$AX = \begin{bmatrix} A_1 & B_1 & & & & \\ C_2 & A_2 & B_2 & & & \\ & \ddots & \ddots & \ddots & & \\ & & C_{p-1} & A_{p-1} & B_{p-1} & \\ & & & C_p & A_p & \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \\ \vdots \\ X_{p-1} \\ X_p \end{bmatrix} = \begin{bmatrix} F_1 \\ F_2 \\ \vdots \\ F_{p-1} \\ F_p \end{bmatrix} = F \tag{8}$$

This partitioning step needs to balance the work-load among processors and should reduce the communication costs. An outstanding option is to consider the well known *chains-on-chains* partitioning [19], together with the *MinMax* algorithm [20]. In summary, the domain decomposition preprocessing stage is defined by the RCM reordering and by the partitioning algorithms (graphic manipuation stages), all

performed before any calculations related to the finite element formulations. We emphasize that this form of domain decomposition is assured whenever the RCM algorithm can reduce the bandwidth of the finite element matrices. Unfortunately, if the RCM algorithm fails, the technique cannot be employed.

Once the domain decomposition preprocess is completed, the narrowband linear system format is adapted to the context of parallel computing in combination with global preconditioners based on Compressed Storage Row (CSR) [5] format or local preconditioners based on element-by-element (EBE) structures [17]. In the most of cases, we consider left preconditioners, only the local Gauss-Seidel preconditioner is a split preconditioner. We point out that the action of all global and local parallel preconditioners is on the matrix-vector products.

3.1 Global Preconditining

One of the most genuine forms to define a preconditioner using CSR storage is to perform the Incomplete LU factorization (ILU) of the original matrix A. We choose to apply the most common sequential incomplete LU factorization, that is, the well-known ILUp [13], where p represents the number of fill-in levels admitted. For each partition i, we neglect the coupling blocks, B_i and C_i, and consider only the sequential ILU factorization of the block A_i (see Eq. 8), as

$$A_i \approx \tilde{L}_i \tilde{U}_i \tag{9}$$

where for each partition i the matrix $M_i = \tilde{L}_i \tilde{U}_i$ and the factors $\tilde{L}_i$ and $\tilde{U}_i$ consider p fill-in levels. This kind of preconditioner is identified as a parallel block Jacobi approach [5], and the factor $\tilde{L}_i$ and $\tilde{U}_i$ are assembled by CSR storage.

For a long time, ILU preconditioners were widely believed to be ill-suited for implementation on parallel computers with more than a few processors. The reason is that the Gaussian elimination, on which ILU techniques are based, offers limited scope for parallelization. However, a series of excellent works about parallel ILU factorizations presented in [3] motivated our parallel domain decomposition version of the ILU preconditioner.

3.2 Local Preconditioning

On the other hand, it is well known that local preconditioners consumed less memory than global preconditioners. Even more, for finite element applications, assembling the entire global matrix can be costly. In this case, the preconditioners construction would be at the element level [21]. Such as in [12], we proposed versions of well known sequential element-by-element preconditioners, as: a diagonal based preconditioners (DIAGe and BlockDIAGe), Gauss-Seidel preconditioners (SGSe and BlockSGSe), and LU factorization preconditioners (LUe and BlockLUe). Prefix "Block" means that the preconditioner was designed for problems with more than one degree of freedom per node. Suffix "e", in turn, denotes that the preconditioner is stored in element-by-element format.

The local preconditioners are based on an abstraction to represent the relations between the local matrices A^e within a partition i, that is,

$$A = \bigcup_{i=1}^{p} A_i^{EBE} \text{ with } A_i^{EBE} = \mathop{\mathbf{A}}_{e=1}^{\mathtt{nel}_i} A^e \tag{10}$$

where, for a specific partition i, A_i^{EBE} is a substitute structure for blocks C_i, A_i, and B_i from Eq. (8); A^e is an elementary finite element matrix of size $\mathtt{ndof} \times 3$ since our formulation admitted only triangular linear elements; $\mathtt{ndof}$ is the degrees of freedom per node and $\mathtt{nel}_i$ is the number of elements of the partition i. In practice, operator $\mathbf{A}$ is just symbolic, since this assembly is never performed when the element-by-element storage scheme is used.

Due to the local preconditioners are based on the diagonal (or block diagonal for problems with $\mathtt{ndof} > 1$) of local matrices A^e, a scaling is used as a pre-preconditioner to normalize the coefficients

of A by its diagonal coefficients (or block diagonal). The original system defined in Eq. (8) may be rewritten as $\tilde{A}X = \tilde{F}$ in which

$$\tilde{A} = \bigcup_{i=1}^{p} \tilde{A}_i^{EBE} \text{ with } \tilde{A}_i^{EBE} = \mathop{\mathbf{A}}_{e=1}^{\mathtt{nel}_i} \tilde{A}^e \tag{11}$$

and

$$\tilde{F} = \bigcup_{i=1}^{p} \tilde{F}_i^{EBE} \text{ with } \tilde{F}_i^{EBE} = \mathop{\mathbf{A}}_{e=1}^{\mathtt{nel}_i} \tilde{F}^e, \tag{12}$$

where $\tilde{A}^e$ and $\tilde{F}^e$ are local scaling of A^e and F^e:

$$\tilde{A}^e = \begin{bmatrix} A_I^{-1}A_{11}^e & A_I^{-1}A_{12}^e & A_I^{-1}A_{13}^e \\ A_J^{-1}A_{21}^e & A_J^{-1}A_{22}^e & A_J^{-1}A_{23}^e \\ A_K^{-1}A_{31}^e & A_K^{-1}A_{32}^e & A_K^{-1}A_{33}^e \end{bmatrix} \tag{13}$$

and

$$\tilde{F}^e = \begin{bmatrix} A_I^{-1}F_1^e \\ A_J^{-1}F_2^e \\ A_K^{-1}F_3^e \end{bmatrix}, \tag{14}$$

with A_{11}^e, A_{12}^e, A_{13}^e, A_{21}^e, A_{22}^e, A_{31}^e, A_{32}^e, and A_{33}^e block matrices of dimension $\mathtt{ndof} \times \mathtt{ndof}$ for each element matrix A^e. The submatrices A_I, A_J, and A_K have dimension $\mathtt{ndof} \times \mathtt{ndof}$ but represent the global block matrices according to the number of degrees of freedom of the global nodes I, J, and K, respectively. F_1^e, F_2^e, and F_3^e, in turn, are blocks of dimension $\mathtt{ndof} \times 1$ for each element vector F^e. The inverse of each global block matrix A_I, with $1 \leq I \leq \mathtt{nnodes}_i$ ($\mathtt{nnodes}_i$ defines the number of nodes in a partition i of the mesh), is calculated explicitly.

REMARK 1: The scaling (or block-scaling) is proposed with exactly 3 subgroups because only triangular linear elements are considered in our finite element formulations.

REMARK 2: To obtain a block A_I, all the contributions of the element matrices that have node I as a common node are added. For example, if a node I is surrounded by 6 elements, the contributions of these 6 element matrices must be considered to compute the block A_I. For simplicity, consider a problem with one degree of freedom per node. In this case, A_I would be a real number represented by the sum of all coefficients of the element matrices that are related to the node I. In the general case, when $\mathtt{ndof} > 1$, A_I will be a matrix block of dimension $\mathtt{ndof} \times \mathtt{ndof}$.

REMARK 3: After to calculate the inverse of the block A_I, it is necessary to check if some row of A_I – or degree of freedom – is associated with Dirichlet boundary conditions. In the affirmative case, the corresponding row and column of A_I^{-1} should be set with the row and column of the equivalent identity matrix.

DIAG*e* and BlockDIAG*e* Preconditioners

The numerical results of the DIAGe and BlockDIAGe preconditioners are equivalent to the scaling processes, as described in Eqs. (11) and (12). That is, such preconditioners are basically a simulation of a scaling. There are two possibilities to perform this issue:

(i) DIAGe and BlockDIAGe are executed just once as a scaling described in Eqs. (11) and (12), exactly to perform the GMRES algorithm on the linear system $AX = F$;

(ii) DIAGe and BlockDIAGe are not executed as described in Eqs. (11) and (12). Instead that, after each matrix-vector product, we perform some convenient scalar multiplications on each resulting vector. The effective preconditioning is a simple multiplication of the diagonal (or block diagonal) over the vector resulting from matrix-vector product.

Despite the preconditioners need to be applied for every iteration when a matrix-vector product is required, the runtime of the DIAGe or BlockDIAGe for the second possibility is smaller when compared to diagonal (or block diagonal) scalings presented by the first possibility. Thus, these preconditioners are set according to the second possibility described. Moreover, these are the only two local preconditioners tested here that need to be assembled for each node.

LU*e* and BlockLU*e* Preconditioners

Let $\bar{A}^e$ be an approximation of each element matrix $\tilde{A}^e$:

$$\bar{A}^e = \begin{bmatrix} I_{\texttt{ndof}} & A^e_{12} & A^e_{13} \\ A^e_{21} & I_{\texttt{ndof}} & A^e_{23} \\ A^e_{31} & A^e_{32} & I_{\texttt{ndof}} \end{bmatrix}, \tag{15}$$

where A^e_{mn}, with $1 \leq m, n \leq 3$, are block matrices of dimension $\texttt{ndof} \times \texttt{ndof}$ for each element matrix $\tilde{A}^e$ and $I_{\texttt{ndof}}$ is the identity matrix of order $\texttt{ndof}$. Preconditioners LUe and BlockLUe are defined simply as LU decompositions of $\bar{A}^e$ in element level. The preconditioning matrix M_i can be defined as

$$M_i = \mathop{\mathbf{A}}_{e=1}^{\texttt{nel}_i} \bar{A}^e = \mathop{\mathbf{A}}_{e=1}^{\texttt{nel}_i} L^e U^e. \tag{16}$$

For each matrix-vector $p_i = M_i^{-1} \tilde{A}_i^{EBE} v_i$, with $\tilde{A}_i^{EBE}$ defined as in the Eq. (11), preconditioners LUe and BlockLUe are applied following two main steps:

Step 1: Perform the matrix-vector product $z_i = \tilde{A}_i^{EBE} v_i$;

Step 2: Calculate the lower and upper triangular systems from $p_i = M_i^{-1} z_i$,

$$p_i = M_i^{-1} z_i \Longrightarrow M_i p_i = z_i \Longrightarrow \mathop{\mathbf{A}}_{e=1}^{\texttt{nel}_i} (L^e U^e p^e = z^e)$$

$$(\texttt{lower triangular}) \Longrightarrow L^e q^e = z^e$$

and

$$(\texttt{upper triangular}) \Longrightarrow U^e p^e = q^e.$$

REMARK 4: The triangular systems $L^e q^e = z^e$ and $U^e p^e = q^e$ are solved for each element. That is, a single loop with $e = \{1, ..., \texttt{nel}\}$ is executed whenever the preconditioner is applied.

SGS*e* and BlockSGS*e* Preconditioners

SGSe and BlockSGSe are split preconditioners given by the preconditioning matrices M_{Li} and M_{Ri}. These matrices are obtained by the trivial Gauss-Seidel decomposition of matrix $\bar{A}^e$ (defined in Eq. 15) at the element level as

$$M_{Li} = \mathop{\mathbf{A}}_{e=1}^{\texttt{nel}_i} L^e \text{ with } L^e = \begin{bmatrix} I_{\texttt{ndof}} & O & O \\ A^e_{21} & I_{\texttt{ndof}} & O \\ A^e_{31} & A^e_{32} & I_{\texttt{ndof}} \end{bmatrix} \tag{17}$$

and

$$M_{Ri} = \overset{1}{\underset{\mathtt{e=nel}_i}{\mathbf{A}}} U^e \text{ with } U_e = \begin{bmatrix} I_{\mathtt{ndof}} & A^e_{12} & A^e_{13} \\ O & I_{\mathtt{ndof}} & A^e_{23} \\ O & O & I_{\mathtt{ndof}} \end{bmatrix}. \tag{18}$$

For each matrix-vector $p_i = M_{L_i}^{-1} \tilde{A}_i^{EBE} M_{R_i}^{-1} v_i$, preconditioners SGS$e$ and BlockSGSe are applied following three main steps:

Step 1: Calculate the upper triangular system from $w_i = M_{R_i}^{-1} v_i$,

$$w_i = M_{R_i}^{-1} v_i \Longrightarrow M_{Ri} w_i = v_i \Longrightarrow \overset{1}{\underset{\mathtt{e=nel}_i}{\mathbf{A}}} \left(U^e w^e = v^e \right);$$

Step 2: Perform the matrix-vector product $z_i = \tilde{A}_i^{EBE} w_i$;

Step 3: Calculate the lower triangular system from $p_i = M_{L_i}^{-1} z_i$,

$$p_i = M_{L_i}^{-1} z_i \Longrightarrow M_{Li} p_i = z_i \Longrightarrow \overset{\mathtt{nel}_i}{\underset{e=1}{\mathbf{A}}} \left(L^e p^e = z^e \right).$$

4 Numerical Experiments

The numerical experiments were performed on the Lobo Carneiro cluster (LoboC)[1] operating with 504 CPUs Intel Xeon E5-2670v3 (Haswell), totalizing 6048 cores. LoboC has 252 processing nodes, and each node has 64GB of RAM and 24 cores (48 with Hyper-Threading). Our codes have been developed in C language and compiled with Intel compiler version 2017.5.239. The parallel environment takes into account the same Intel version of the Message Passing Interface (MPI) protocol. Speedup is redefined proportionally to 24 MPI ranks,

$$Speedup(n) \quad = \quad \frac{CPU\ time\ of\ 24\ MPI\ ranks}{CPU\ time\ of\ n\ MPI\ ranks}, \tag{19}$$

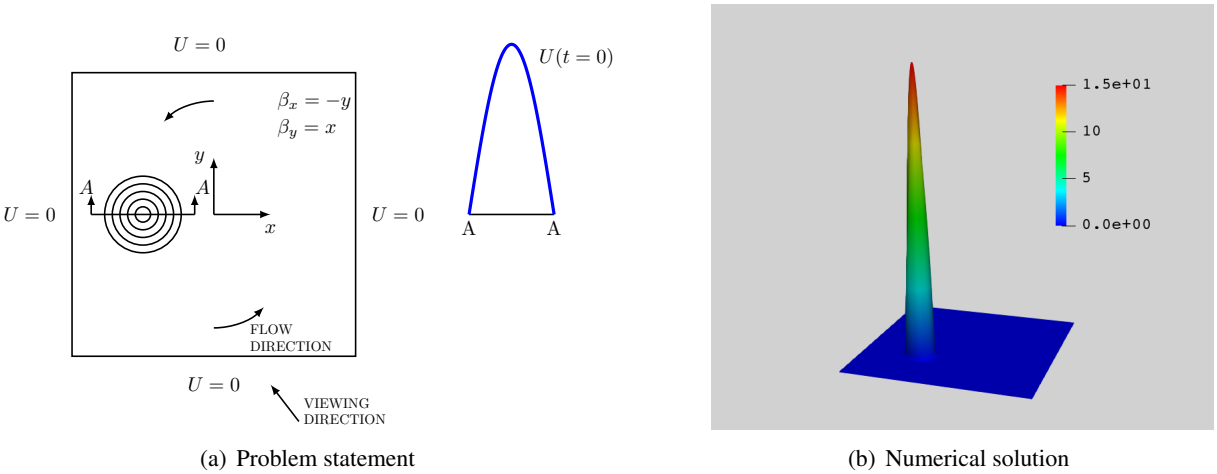
and preconditioner memory usage is evaluated using the Valgrind package[2] with the Massif[3] tool. Here, we evaluate the most significant preconditioner functions: $Build\_Matrices$ obtains the finite element matrices and independent vectors; $Matrix\_Vector\_Product$ process the matrix-vector products; $Preconditioner\_Setup$ prepares the preconditioning structures; $Preconditioner$ is in charge of the preconditioning itself; $Vectors$ is formed by the vector operations as implemented in BLAS level 1; $MPI\_Allreduce$, $MPI\_Barrier$, MPI_Isend, and MPI_Recv are the native MPI functions. Since the linear systems size and preconditioning can vary significantly for problems with different number of degrees of freedom per node, we consider two representative benchmarks: the rotating fluid flow field modeled by transport equation (1 degree of freedom per node) and the explosion problem modeled by the Euler equations (4 degrees of freedom per node). In both cases, a parallel preconditioned GMRES solver is used with tolerance $\varepsilon_{GMRES} = 10^{-6}$ and 30 Krylov basis vectors.

4.1 Rotating cone – Modeled by Transport Equation

A rigid rotation of a cone about the center of the square domain $\Omega = [0, 10] \times [0, 10]$ is the transient problem considered, as described in [22]. Figure 1(a) shows the problem statement, where the diffusivity

[1] `http://portal.nacad.ufrj.br/recurso-icex.html`
[2] `http://valgrind.org/`
[3] `http://valgrind.org/info/tools.html#massif`

is $\epsilon = 10^{-8}$, the flow velocity is $\beta = (-y, x)^T$, and the reaction coefficient and the source term are null in Eq. (2). We use a time step size $\Delta t = 10^{-2}$ in the interval $[0, 6.28]$, and 3 multi-corrections on each timestep. Consequently, the parallel preconditioned GMRES method runs exactly 1884 times to solve this problem. Figure 1(b) presents the transient solution of the rotating cone at time $t = 6.28$ seconds, using ILU0 in 24 MPI ranks. A mesh with 1,321,646 nodes and 2,639,290 triangular linear elements and



(a) Problem statement

(b) Numerical solution

Figure 1. Rotating cone: (a) problem statement; (b) solution with ILU0 in 24 MPI ranks.

six preconditioning cases are considered: three local preconditioners, DIAGe, LUe and SGSe, and three global preconditioners, ILU0, ILU1 and ILU2. In this example, the bandwidth is reduced from 1,317,171 to 2,077 in 3.8 seconds using the RCM algorithm. The total time to perform the domain decomposition preprocess is less than 37 seconds and the memory usage is 843 MB.

This parallel experiment uses 24, 48, 72, 96, and 192 MPI ranks. Table 1 presents total CPU time and average number of GMRES iterations per GMRES execution (iter$_{av}$) – the total number of GMRES iterations is divided by 1884. DIAGe, LUe and SGSe preconditioners keep their iter$_{av}$ almost constant when the number of MPI ranks increases. The ILUp preconditioner with the lower fill-in, ILU0, is the best option among all global preconditioners. However, comparing all preconditioners (local and global), LUe presents the best results in terms of iter$_{av}$ and CPU time.

Table 1. Rotating cone: CPU time and the average number of GMRES iterations.

Type	Number of MPI ranks									
	24		48		72		96		192	
	time	iter$_{av}$	time	iter$_{av}$	time	iter$_{av}$	time	iter$_{av}$	time	iter$_{av}$
DIAGe	1157.2	65.8	559.8	64.3	388.1	64.2	285.9	65.0	133.6	65.5
LUe	355.1	11.5	178.5	11.6	123.1	11.6	89.8	11.6	43.3	12.0
SGSe	771.6	24.5	380.9	24.7	252.1	24.6	179.4	24.5	72.1	24.4
ILU0	372.6	12.8	186.1	14.8	119.6	16.0	84.7	17.0	44.0	20.5
ILU1	375.4	12.4	190.5	14.3	124.2	15.5	91.8	16.8	45.2	20.1
ILU2	420.9	12.4	212.6	14.3	142.6	15.5	105.0	16.8	50.7	20.3

Figures 2 shows CPU time and speedup for all MPI ranks. SGSe presents superlinear speedup. However, LUe is about three times faster than DIAGe and twice as fast as SGSe. ILU0 and ILU1 present runtime similar to LUe. All preconditioners present good scalability. The graphics of functions runtime

analysis for all parallel are showed in Fig. 3. In general, ILU preconditioners demand similar runtimes to perform functions *Build_Matrices*, *Matrix_Vector_Product*, and *Vector*, since they presented similar average number of GMRES iterations (see Tab. 1). Local preconditioners *Build_Matrices* runtime is slightly smaller when compared with corresponding global preconditioners functions. That occurs due to complexity of CSR matrix assembly. On the other hand, local preconditioners *Matrix_Vector_Product* runtimes are the largest. Even so, LUe is the fastest because *Preconditioner* and *Precontitioner_Setup* are much smaller compared to the corresponding ILUp functions (fact assured by the smaller number of GMRES iterations obtained by LUe – see Tab. 1). We also note that the CPU time demanded by $MPI\_Allreduce$ is proportionally larger when the number of MPI ranks is greater.

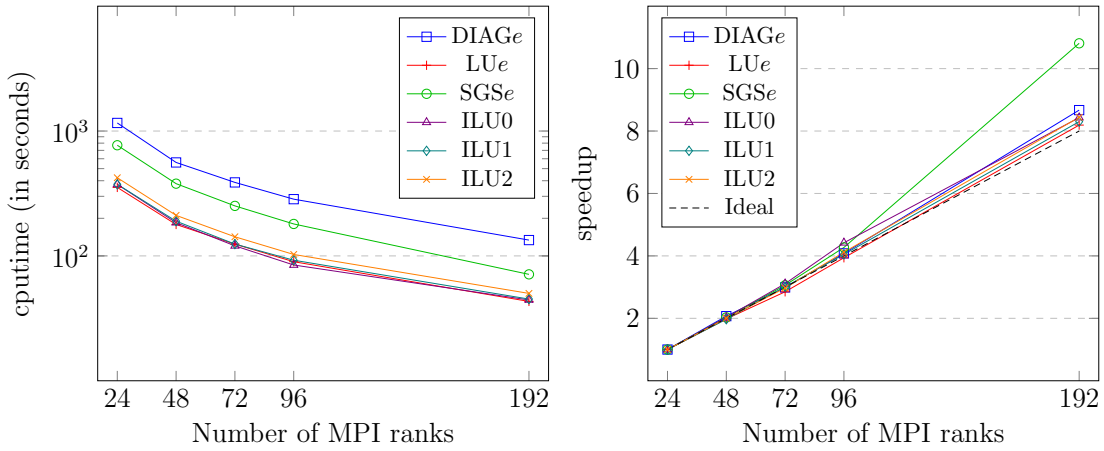


Figure 2. Rotating cone – CPU time (left) and speedup (right).

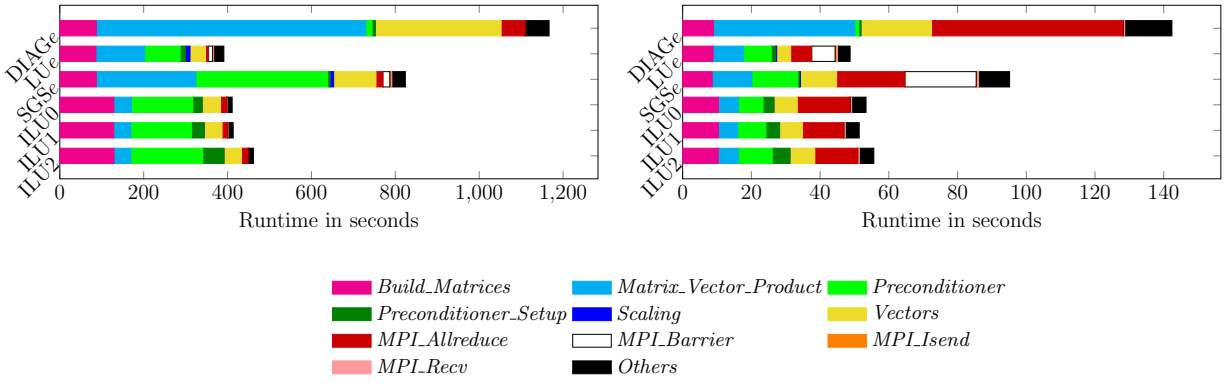
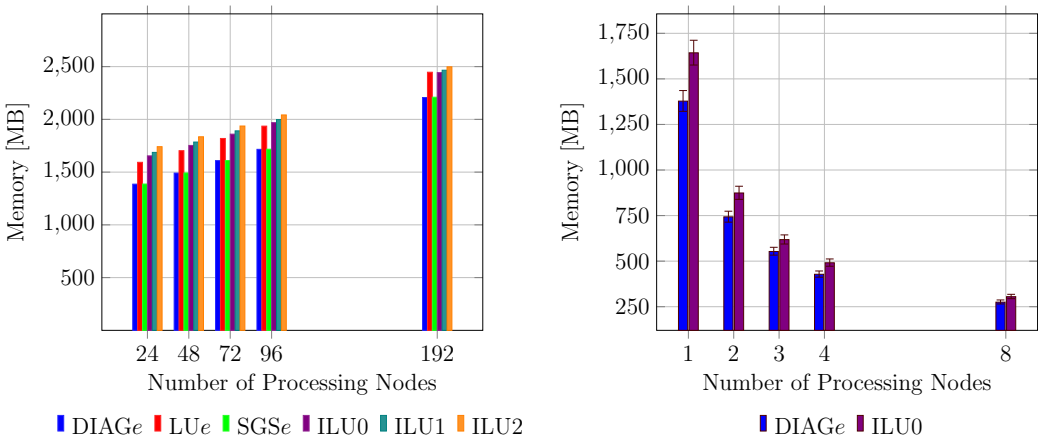


Figure 3. Rotating cone – Analysis for the functions runtime - (left 24, right 192).

Figure 4(a) shows the graphics of total memory usage for all parallel preconditioners and number of MPI ranks. ILU preconditioners demand more memory compared with local preconditioners and when the fill-in level increases. On the other hand, LUe demands more memory when compared with the other local preconditioners. The average memory usage for DIAGe and ILU0 preconditioner considering 1, 2, 3, 4, and 8 processing nodes are showed in Fig. 4(b). There is not a very large discrepancy between the minimum and maximum memory values used in each MPI rank. This behavior confirms that our domain decomposition approach Lima et al. [9] has been successful in load balancing for global and local preconditioners for problems modeled by transport equation.

4.2 Explosion Problem – Modeled by Euler Equations

The second problem considered is a transient problem with 4 degrees of freedom per node. That problem, known as explosion problem, is described in Toro [23]. The 2D Euler equations are solved in



(a) Total (b) Average per node

Figure 4. Rotating cone – Memory usage.

a 2.0×2.0 square domain in the $xy-$plane. The initial condition consists of the region inside of a circle with radius $R = 0.4$ centered at $(1, 1)$ and the region outside the circle, see Fig. 5. The flow variables are constant in each of these regions and are separated by a circular discontinuity at time $t = 0$. The two constant states are chosen as

$$\text{ins} \begin{cases} \rho &= 1.0 \\ u &= 0.0 \\ v &= 0.0 \\ p &= 1.0 \end{cases} \qquad \text{out} \begin{cases} \rho &= 0.125 \\ u &= 0.0 \\ v &= 0.0 \\ p &= 0.1 \end{cases} \tag{20}$$

The time advancing is solved using the parallel predictor-multi-corrector algorithm as described in the Algorithm 1. The final time $t_{final} = 0.25$ and the time step $\Delta t = 10^{-3}$ are chosen to represent the numerical solution. For each time step is set 3 fixed multi-corrections. In short, the parallel preconditioned GMRES runs exactly 750 times to solve each experiment.

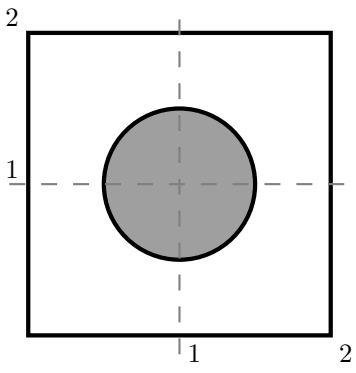


Figure 5. Explosion: Problem statement.

An unstructured mesh with 531,166 nodes and 1,059,798 triangular linear elements and five preconditioning cases are considered: the three local preconditioners (BlockDIAGe, BlockLUe, and BlockSGSe) and ILU0, ILU1 global preconditioners. In this example, the bandwidth is reduced from 2,114,275 to 2,663 in 6.1 seconds using the RCM algorithm. The total time to perform the domain decomposition preprocess is less than 55 seconds and the memory used is 2094 MB. As we can highlight, the number

of unknowns is about four times greater than the number of nodes because it is a problem with 4 degrees of freedom per node. This experiment consider 24, 96, 192, and 384 MPI ranks.

Figure 6 presents the density solution of the explosion problem at time $t = 0.25$ second, obtained with the preconditioner ILU0 and 24 MPI ranks.

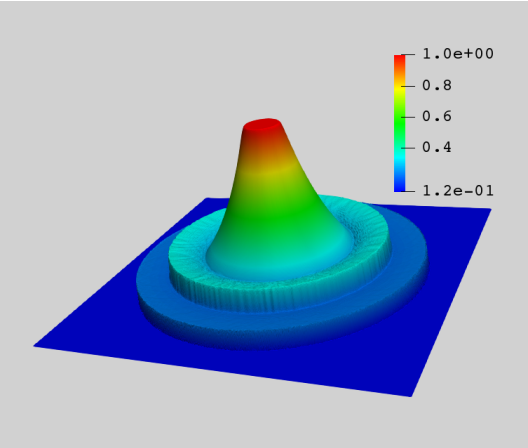


Figure 6. Problem 3 : Density solution at $t = 0.25$ second with the preconditioner ILU0 and 24 MPI ranks.

Table 2 presents the total CPU time and average number of GMRES iterations per GMRES execution (iter$_{av}$) – the total number of GMRES iterations is divided by 750. This table shows, respectively, the parallel preconditioners BlockDIAGe, BlockLUe, BlockSGSe, ILU0, and ILU1 considering 24, 96, 192, and 384 MPI ranks. As we can note, iter$_{av}$ performed by the BlockDIAGe preconditioner is basically constant. Such behavior is expected since this preconditioner is just a block scaling based on the main block diagonal of the matrix (see Section 3.2) and it has the same effect of preconditioning independent of the number of MPI ranks. Local preconditioners BlockLUe and BlockSGSe present a number of GMRES iterations (iter$_{av}$) much smaller than that reached by the preconditioner BlockDIAGe, however, the CPU times are larger. Unfortunately, that becomes BlockLUe and BlockSGSe less attractive than BlockDIAGe. For a number of 24 and 96 MPI ranks, BlockDIAGe presented the best CPU times at all, however, from 192 MPI ranks, preconditioner ILU0 became more advantageous.

Table 2. Explosion problem: CPU time and the average number of GMRES iterations.

Type	Number of MPI ranks							
	24		96		192		384	
	time	iter$_{av}$	time	iter$_{av}$	time	iter$_{av}$	time	iter$_{av}$
BlockDIAGe	334.5	13.7	83.8	13.7	49.8	13.7	35.3	13.7
BlockLUe	428.4	8.2	110.8	8.4	65.9	8.4	46.5	8.7
BlockSGSe	442.3	7.5	112.8	7.5	67.18	7.5	45.4	7.6
ILU0	382.2	7.4	98.9	8.6	47.6	9.3	23.4	10.5
ILU1	453.4	7.4	115.9	8.6	56.5	9.3	27.0	10.5

Figure 7 shows CPU time and speedup for local and global preconditioners. BlockDIAGe, BlockLUe, and BlockSGSe present scalability until 192 partitions. However, for 384 partitions, the speedup dropped significantly. Such behavior is related to the fact that EBE matrix-vector products for problems with `ndof` > 1 tend to suffer more with the overlapping generated by the greater number of partitions. As can

be emphasized, BlockLUe and BlockSGSe reduced the number of iterations, but the performance is not substantial when compared with the preconditioner BlockDIAGe. On the other hand, speedup of ILU preconditioners point out good scalability.

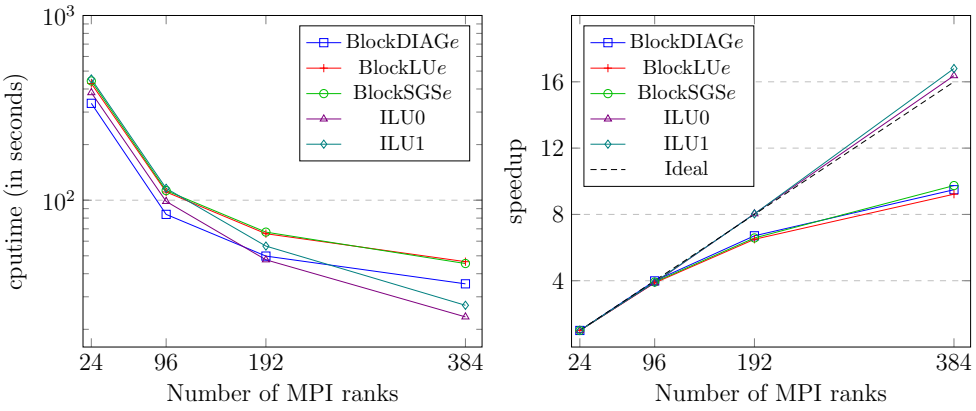


Figure 7. Explosion problem – CPU time (left) and speedup (right).

Figure 8 shows the functions runtime analysis for the parallel preconditioners BlockDIAGe, BlockLUe, BlockSGSe, ILU0, and ILU1 considering 24 and 384 MPI ranks. When 24 MPI ranks are considered, BlockDIAGe present the best overall runtime even its $Matrix\_Vector\_Product$ demands the greatest runtime. As the number of MPI ranks increases, ILU preconditioners become the best options, because the runtimes of the $Preconditioner$ and $Preconditioner\_Setup$ ILU functions decrease significatively. Function $Build\_Matrices$ keeps similar behavior as the problem modeled by transport equation – $Build\_Matrices$ is slower for global preconditioners because CSR assembly is more complex. Note the $Matrix\_Vector\_Product$ function demands a greater runtime for local preconditioners, i.e., in our implementation, CSR matrix-vector products are more efficient than EBE ones. The runtime of the functions $Scaling$, $MPI\_Barrier$ and even of the functions $Preconditioner$ and $Preconditioner\_Setup$ make these local preconditioners a bad choice to solve this problem. As in the previous experiments, $MPI\_Allreduce$ runtime is proportionally larger for a high number of MPI ranks. Runtimes of MPI_Isend and MPI_Recv can also be considered proportionally tiny.

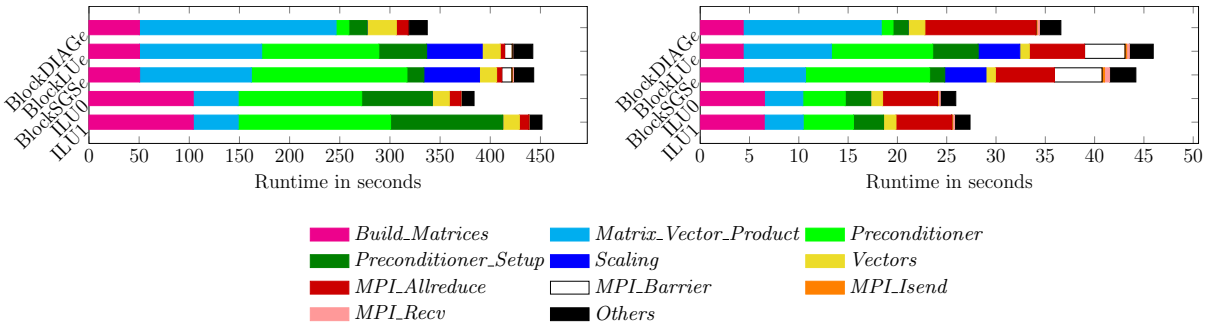
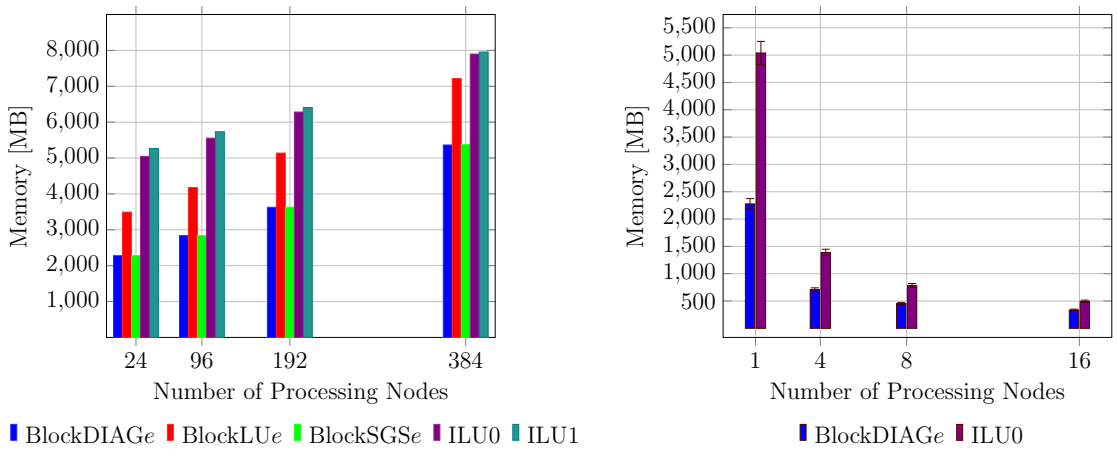


Figure 8. Explosion problem – Analysis for the functions runtime - (left 24, right 384).

Figure 9(a) shows the total memory usage for the parallel preconditioners BlockDIAGe, BlockLUe, BlockSGSe, ILU0, and ILU1 considering 24, 96, 192, and 384 MPI ranks. The difference between the total memory usage of global preconditioners and the total memory usage of local preconditioners becomes more evident. Global preconditioners use more memory during processing, however, this difference tends to decrease as the number of MPI ranks increases.

Figure 9(b) shows the bar min and max graph of average memory usage per node for the local preconditioner BlockDIAGe and the global preconditioner ILU0 considering 1, 4, 8, and 16 processing nodes. In this experiment, there is no large discrepancy between the minimum and maximum memory

values used in each MPI rank as well. We also point out that memory usage of the local preconditioners is significantly lower compared to global preconditioners memory usage. However, as we increase the number of processing nodes this difference is reduced considerably.



(a) Total

(b) Average per node

Figure 9. Explosion problem – Memory usage.

5 Conclusions

In this work, we compare the efficiency of the global parallel ILUp preconditioners, $p = 0, 1, 2$, with the following local parallel preconditioners: diagonal (DIAGe), block-diagonal (BlockDIAGe), LU-factorization (LUe), block-LU-factorization (BlockLUe), Gauss-Seidel (SGSe) and block-Gauss-Seidel (BlockSGSe). The global and local preconditioners are based on the Compressed Storage Row (CSR) format and the element-by-element (EBE) structures, respectively. The robustness and scalability of these parallel preconditioning strategies have been demonstrated for two benchmarks: a rotating cone fluid flow problem modeled by transport equation and the explosion problem modeled by the Euler equations.

In both examples, the bandwidth reduction using the RCM algorithm is very large and also very fast, and the total time to perform the domain decomposition preprocess is very small. Also, the local preconditioners demand less memory usage when compared with the global representatives. Moreover, for the global ILUp preconditioners, higher fill-in levels and, consequently, greater memory consumptions do not mean improvement on CPU time. Instead of that, the best results are reached by the global ILU0 preconditioner. So, in the Euler equations problem we do not consider the ILU2 preconditioner.

For the transport equation problem, all global and local preconditioners present good scalability. In terms of total memory usage, CPU time and average number of GMRES iterations, the local LUe and the global ILU0 preconditioners present the best results. Overall in this example, the local LUe preconditioner is slightly better than the global ILU0 preconditioner. However, the other two local preconditioners (DIAGe e BlockLUe) do not perform well compared with all global ILUp preconditioners tested here. We also observe that the ILU preconditioners demand similar runtimes to perform the most significant preconditioner functions, since the global preconditioners present similar average number of GMRES iterations. This is not the case of the local preconditioners, where the matrix-vector products can be very expensive and the native MPI functions can be proportionally larger when the number of MPI ranks increases.

For the Euler equations problem, the speedups of the ILUp preconditioners point out good scalability. In contrast, the speedups of the local preconditioners dropped significantly for the largest number of MPI ranks. Such behavior is related to the fact that EBE matrix-vector products, for problems with more than 1 degree of freedom per node, tend to suffer more with the overlapping generated by the

greater number of partitions. In this example, the local BlockDIAGe preconditioner presents favorable CPU time and memory usage, but the global ILU0 preconditioner becomes more advantageous when the number of MPI ranks increases. Moreover, the average number of GMRES iterations performed by all local preconditioners is basically constant for all partitions. However, the runtimes of their MPI functions increase significantly. For the global preconditioners, the functions in charge of the preconditioning itself and their structures decrease remarkably. As a consequence, the global ILU0 preconditioner become the best option among all global and local preconditioners. Even more, we also observe in this example the local preconditioners dependence on the number of degrees of freedom per node.

Acknowledgements

The authors would like to thank CAPES by partial financial support and NACAD/UFRJ by use of the LoboC cluster.

References

[1] Van der Vorst, H. A., 2003. *Iterative Krylov methods for large linear systems*, volume 13. Cambridge University Press.

[2] Gutknecht, M. H., 2007. A brief introduction to krylov space methods for solving linear systems. In *Frontiers of Computational Science*, pp. 53–62. Springer.

[3] Wathen, A. J., 2015. Preconditioning. *Acta Numerica*, vol. 24, pp. 329–376.

[4] Saad, Y. & Schultz, M. H., 1986. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal on Scientific and Statistical Computing*, vol. 7, n. 3, pp. 856–869.

[5] Saad, Y., 2003. *Iterative methods for sparse linear systems*, volume 82. SIAM.

[6] Manguoglu, M., Koyutürk, M., Sameh, A. H., & Grama, A., 2010. Weighted matrix ordering and parallel banded preconditioners for iterative linear system solvers. *SIAM Journal on Scientific Computing*, vol. 32, n. 3, pp. 1201–1216.

[7] Arbenz, P. & Gander, W., 1994. A survey of direct parallel algorithms for banded linear systems. *ETH, Eidgenössische Technische Hochschule Zürich, Departement Informatik, Institut für Wissenschaftliches Rechnen*, vol. 221.

[8] Polizzi, E. & Sameh, A. H., 2006. A parallel hybrid banded system solver: the spike algorithm. *Parallel Computing*, vol. 32, n. 2, pp. 177–194.

[9] Lima, L. M., Lugon, B. A., & Catabriga, L., 2016. An alternative approach of the spike preconditioner for finite element analysis. In *High Performance Computing (HiPC), 2016 IEEE 23rd International Conference on*, pp. 331–340. IEEE.

[10] Lima, L. M., Catabriga, L., Rangel, M. C., & Boeres, M. C. S., 2017. A trade-off analysis of the parallel hybrid spike preconditioner in a unique multi-core computer. In *International Conference on Computational Science and Its Applications*, pp. 422–437. Springer.

[11] Liu, W.-H. & Sherman, A. H., 1976. Comparative analysis of the Cuthill–McKee and the reverse Cuthill–McKee ordering algorithms for sparse matrices. *SIAM Journal on Numerical Analysis*, vol. 13, n. 2, pp. 198–213.

[12] Muller, L. K., Lima, L. M., & Catabriga, L., 2017. A comparative study of local and global preconditioners for finite element analysis. In *Proceedings of the XXXVIII Iberian Latin-American Congress on Computational Methods in Engineering*.

[13] Meijerink, J. A. & van der Vorst, H. A., 1977. An iterative solution method for linear systems of which the coefficient matrix is a symmetric m-matrix. *Mathematics of Computation*, vol. 31, n. 137, pp. 148–162.

[14] Brooks, A. N. & Hughes, T. J., 1982. Streamline Upwind/Petrov-Galerkin formulations for convection dominated flows with particular emphasis on the incompressible Navier-Stokes equations. *Computer Methods in Applied Mechanics and Engineering*, vol. 32, n. 1-3, pp. 199–259.

[15] Bazilevs, Y., Calo, V. M., Tezduyar, T. E., & Hughes, T. J., 2007. YZβ discontinuity capturing for advection-dominated processes with application to arterial drug delivery. *International Journal for Numerical Methods in Fluids*, vol. 54, n. 6-8, pp. 593–608.

[16] Tezduyar, T. E. & Senga, M., 2006. Stabilization and shock-capturing parameters in supg formulation of compressible flows. *Computer Methods in Applied Mechanics and Engineering*, vol. 195, n. 13-16, pp. 1621–1632.

[17] Hughes, T. J., 2012. *The finite element method: linear static and dynamic finite element analysis*. Courier Corporation.

[18] Lima, L. M., 2018. *An alternative approach to parallel preconditioning for 2D finite element problems*. PhD thesis, Universidade Federal do Espírito Santo, Vitória, ES, Brazil.

[19] Pınar, A. & Aykanat, C., 2004. Fast optimal load balancing algorithms for 1d partitioning. *Journal of Parallel and Distributed Computing*, vol. 64, n. 8, pp. 974–996.

[20] Manne, F. & Sorevik, T., 1995. Optimal partitioning of sequences. *Journal of Algorithms*, vol. 19, n. 2, pp. 235–249.

[21] Hughes, T. J., Levit, I., & Winget, J., 1983. An element-by-element solution algorithm for problems of structural and solid mechanics. *Computer Methods in Applied Mechanics and Engineering*, vol. 36, n. 2, pp. 241–254.

[22] Hughes, T. J. & Tezduyar, T., 1984. Finite element methods for first-order hyperbolic systems with particular emphasis on the compressible euler equations. *Computer Methods in Applied Mechanics and Engineering*, vol. 45, n. 1-3, pp. 217–284.

[23] Toro, E. F., 2013. *Riemann solvers and numerical methods for fluid dynamics: a practical introduction*. Springer Science & Business Media.